International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 09 Issue: 02 | Feb 2022 www.irjet.net p-ISSN: 2395-0072

Limited Budget but Effective End to End MLOps Practices (Machine Learning Model Operationization) in Generic On-Premises Platform

Indranil Dutta

Principal Consultant and Lead Data Scientist

Abstract : *In this research, would like to bring a mechanism to implement a typical MLops pipeline for small scale organization who cannot afford the operational expenditures to bring the pipeline at Cloudera, Horton works platform or cloud premises like AWS, GCP or Azure. This paper gives a very detailed understanding of operationalization of a typical ML pipelines to adhere all the elements and artifacts without even using any Docker, Kubernetes or even any API generating platforms like Flask or FastAPI. Using the combination of a simple Python/R along with SQL and Shell scripts we can manage the entire workflow at on premises with a very low-cost approach. From some angle this mechanism would not be comparable with the architectures like market ready MLops platforms like Azure Devops, MLflow, Kubeflow, Apache Airflow, Databricks with Data factory or Sagemaker Studio workflow but from conceptual point of view, suffice almost 90% of the requirements with efficient manner.*

Key Words: MLops, ML pipeline, On premises, Low Budget Architecture, ML Engineering

1.INTRODUCTION

In current Data science and Analytics business landscape MLops is a very prominent and fascinating concept that most of the organizations are very optimistic to implement as a part of their AIML initiatives. Different level of organizations across different lines of business has their own policy to adopt the MLops to orchestrate the workflow. Large scale organizations can easily afford expensive licenses or operational expenses to bring their entire ML pipeline in customized platforms like Cloudera Manager CDP, CML or SDX or using the Cloud PaaS or IaaS. There are gamut's of state-of-the-art options to consume the developed ML pipelines and push the same in real production environment and post deployment monitoring and tracking without many difficulties of creating self-made architectures and arrangements.

But if we see the other angles of the industry, there are lot of small players in the market or start-up organization who are nurturing their end-to-end journey with custom architectures and arrangements and without affording the expensive SOTA platforms for operationalizing their ML pipelines.

2. BUSINESS CONTEXT

Consider a typical MLops journey and why its required to implement for a successful business solution. To answer the points lets understand the context of the following questions?

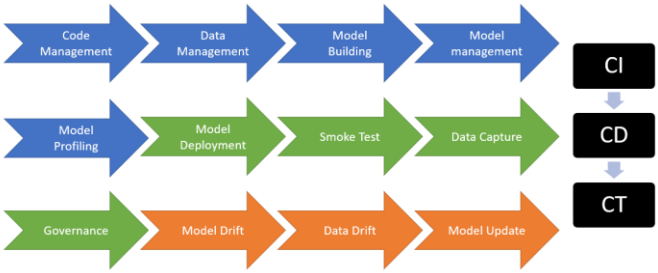
- Where do we capture the data and does the single version of data always suffices to meet the requirement?
- How do we manage the different versions of codes that all analyst/Data scientist creates for different iterations?
- How do we capture the Model artifacts for so many numbers of iterations and a reference point for future comparison?
- How do we capture the model logs to track the performance and deployment status?
- What we do if the model performance degrades after few iterations post deployment?
- How do we monitor the performance of the model in case there are a few lines up for the same utilization?
- How do we profile the Model infrastructure for a seamless production execution based upon the business requirement?
- How do we capture the test iterations results?
- How many times we can manually changes the pipelines and execute them once there are minor/major changes in the code?
- Who will take the initiative to maintain the release of the pipelines in case there are some alterations, and the latest version should be in place?

All these points are very much relevant in today's AIML landscape to successfully implement and maintain a solution in production environment. In the next section onwards we will see how we can leverage the same requirements with some hands on custom architectures, its benefits, limitations and few sample use cases that we can afford the structure

and some sample cases where the solution is unable to provide a very elastic scalable support towards the requirements.

3. ELEMENTS OF A TYPICAL MLOPS PIPELINE

Figure -1: A typical MLOps flow



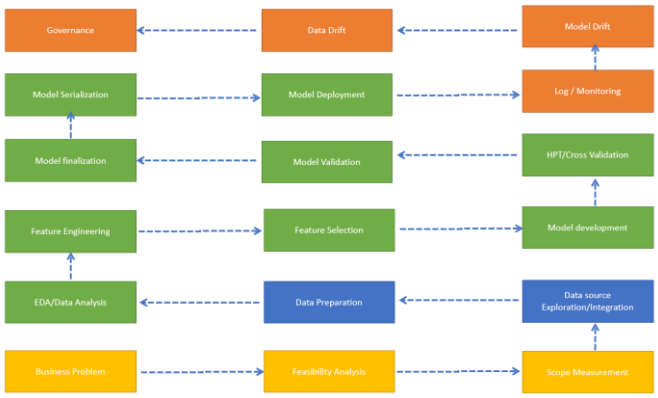
A typical MLOps flow comprise the above-mentioned steps and here are the details.

- **Code Management:** Code versioning and maintenance
- **Data Management:** Versioning, Incremental or Full Load, Central Location
- **Model Building:** Log All metrices, Logs and artifacts, organizing Multiple iterations, Central locations, Meta data and Data Lineage, Provisioning Infrastructure
- **Model Management:** Central Location, Versioning, Logs, Outputs, Hyper Parameters, plots, Meta Data
- **Model Profiling:** Memory and processor requirements.
- **Model Deployment:** Automated deployment, provisioning infrastructure, End point versioning, Traffic Management (scale up and scale out)
- **Smoke Test:** End Point Health, Positive and Negative Unit tests
- **Data Capture:** Real time input and prediction capture, Central location and time capture
- **Governance:** Model approve/Rejection, Notifications
- **Model Drift:** fluctuation on expected Model outcome
- **Data Drift:** Unexpected feature behavior change
- **Model Update:** Keep the model update, retraining, versioning based on latest dataset and code integration.

The entire flow is also embedded with three promising concepts like Continuous Integration (CI), Continuous Deployment (CD) and Continuous training (CT) pipelines. These is the standard structure of a desired MLops structure and now lest explain how these requirements can be served using a custom code-based architecture and how much we can achieve without using any state-of-the-art architectures.

4. SOLUTION DESIGN

Figure -2: A generic flow of a AIML solution



Without thinking from a typical MLops cycle, these are the standard requirements to serve a successful ML operational pipeline which comprises the following steps:

- Identifying Business Problem
- Feasibility Analysis of the problem for any AIML solution
- Scope Measurement and estimation
- Data Source exploration and Integration
- Data Preparation and Data engineering
- Exploratory data Analysis and visualization
- Feature Engineering and exploration
- Feature selection and finalization
- Model building, training, and iterations
- Hyper Parameter turning and optimization
- Cross Validation
- Model validation and testing
- Model Finalization
- Model Serialization

- Model Deployment
- Log Maintenance and Monitoring
- Model drift calculation
- Data drift calculation
- Governance of the entire pipeline and notification

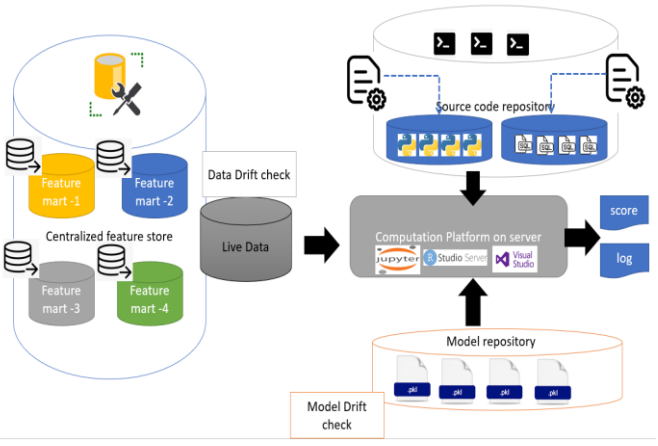
Along with the basic steps also the following needs to be taken care also

- How the code changes effects on the code repository and model release (final one)
- Automating the pipeline so it can consume the live data on each iteration and can complete the require performance.
- Continuous comparing the model performance and incase of any significant deviation trigger the continuous training pipeline to retrain the model and the fresh one would place for next iteration and the same process will keep going on.

5. CORE CONCEPT

Now let's discuss the core operational concept of the discussed solution in details

Figure -3: A typical Model operationalization workflow



Few of the core components of the architecture and its application:

Centralized Feature Repository: There is a single data repository which also comprises multiple Feature marts that also comprises different work profiles and, in a nutshell, these are the unique source of incremental load where the multiple feature marts combined to generate the live data for the model consumption. There would be a feature of data drift calculation that we will discuss in later phase. For future reference also the repository maintains the data versioning which used for each iteration.

NOTE: The entire flow is describing from a single data scientist experiment point of view for the simplicity. But the same structure can handle the same workload for multiple data scientist as well based on the server space and capacity.

Model repository: This is a centralized repository of all the model files that are generated by the data scientists and based on the version number the required one can be accessed from the repository. This is also a serve-based repository and the no of model and its size also depends upon the server space and capacity.

Source Code repository: All the required codes like data preparation and transformation SQL codes, Model execution or scoring codes on Python or R, dependency codes and config files, Shell scripts that also merge few of the transformation and model execution codes in a unique flow. In a nutshell this is a centralized location to cater all the source codes individually or through a pipeline.

Computation Platform: This is the engine where all the codes will execute, it could be a UNIX based platform or an Anaconda environment or a RStudio server. The processing capacity would be defined and varied based upon the serve memory and space allotted for the specific job.

Continuous Integration: There would be separate config files like. YAML which caters all the parameters details and in the release codes will be parametric and call the config files during the execution. Data scientist will only make the changes in the config files and in subsequent execution the changes will reflect in the base code. Hence in a very manual way the integration can be taken place.

Continuous Deployment: There would be the shell scripts which keep executing the pipelines through a CRON scheduler on periodic basis. Whenever the .YAML files uploads the changes will automatically updates in the base code and during the .sh pipeline execution the change will hit the model files. Hence, we are not configuring any trigger based continuous deployment, but this is a periodic deployment to keep the model files updated with respect to any changes in the base code.

Continuous Training: This concept will also cover separately in a different section but already taken care in this architecture.

Model Drift: This is a embedded part of the Continuous training pipeline and also be explained in a subsequent section.

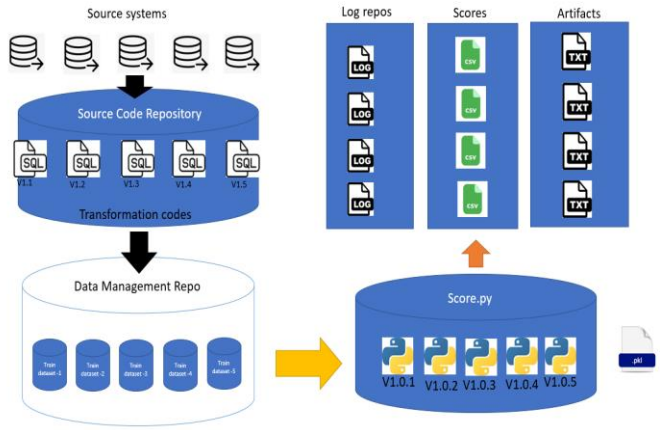
Data Drift: This is also an individual topic that we will discuss in a separate section.

Governance: The logging mechanism will be there to monitor and notify against all the key performance and selection/rejection of the specific version of the models and the final acceptance criteria.

Visualization/Result or Drift showcasing: For this purpose, we are not using any third-party tools like Tableau or power BI. In python own capability we can use Dtale or Sweetviz to create interactive visualization within Jupyter environment and can drag and drop features to showcase a presentable outcome.

The flow is here depicting that the core computational platform will call the latest model files based on the user's discretion and the specific code version from the code repository which is already integrated. Finally, it consumes the live data block from the unified feature store and then complete the execution in the platform and generates the output file through a csv, a log file and the model artifacts and relevant metrices with .txt file which will store in a model output repository for future references.

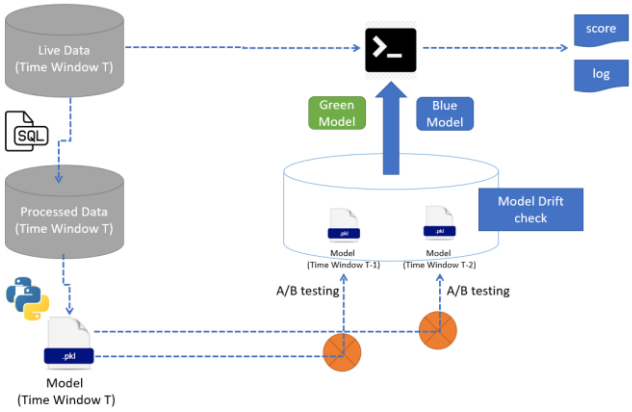
Figure -4: An execution workflow



This is in a nutshell a snapshot of the model execution workflow which covers the versioning and end to end integration and entire pipeline would be embedded through a shell script and the same will scheduled through a cron for periodic execution based upon the business requirement.

6. MODEL DRIFT & CONTINOUS TRAINING PIPELINE

Figure -5: CT pipeline with model drift calculation



This is a very crucial steps and would like to elaborate step by step.

1. From the data repository (unique feature store the live data block would be ready for the current timestamp – Window T)

2. Using the sql transformation the processed data is ready for the model consumption, that is also for the same time window T.

3. Now the shell script will make the execution and compare the results with the benchmark or with the result of last execution (T-1) and (T-2). In case there is any significant changes we call this as model drift.

4. This triggers another pipeline were consuming the processed data at Time window T a retrainable python script will execute and generate a new model version (Time window T).

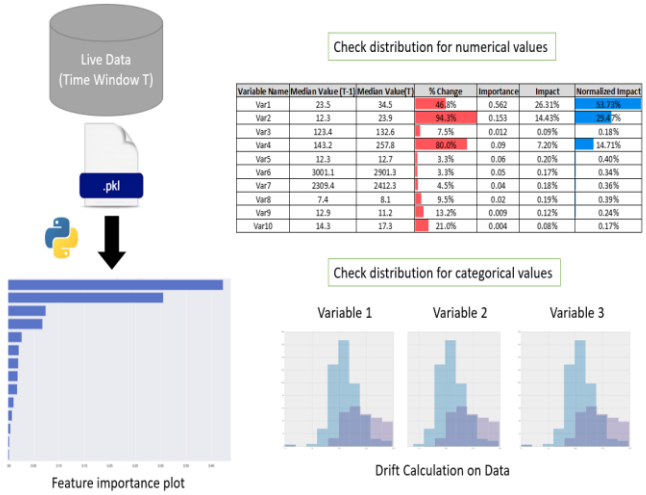
5. Now compare the KPIs of the model of Time window T along with the model of Time window (T-1) and (T-2) and then decide which one would be productionized at that moment. This type of model deployment is called a BLUE GREEN deployment and finally based upon the comparison (A/B testing) it's been decided whether we will go for the blue model or Green one.

6. Then the final model would deploy on server and the subsequent scoring will happen from these models itself.

This is how the model drift monitoring will happen along with the continuous training. This is a continuous iterative process and whenever there would be a significant drift and new live data block will hit the pipeline the same exercise will happen and update the model in the target sever.

7. DATA DRIFT & IMPACT ANALYSIS

Figure -6: Data drift engagement



This is a very interesting solution to identify and visualize the data drift and subsequent impact analysis.

Here the final training will generate the model file and the list of important parameters along with degree of importance.

Now once there would be significant drift in the data for those important parameters then only this will impact the model performance. In case there are huge drift with some parameters which is not at all a driver parameter for the model will not cause much fluctuation in the model performance.

Now there are generally two different types of parameter – numeric and categorical. For categorical we will simply check the distribution of the lagged version and the current version and interpret the same visually using the Pandas profiling, SweetViz or Dtale platform.

For the numeric variables' median is a better statistical benchmark and we will compare the same for the latest data with respect to the last version data and plot the drift based on the distribution median of the numeric values. Now we will multiple the model importance ratio (normalized one) along with the drift parameters and calculate the impact.

Thereby its very robust solution to estimate the impact caused by each parameters drift and the same can be logged for all iterations for future references.

8. SAMPLE USE CASE FITS INTHIS ARCHITECTURE

There are few sample traditional AIML use case that fits in this stricture and performs well:

- Customer Churn propensity Model
- Marketing Mix Model
- Customer Lifetime Value
- Credit Scoring Model for Banking
- Upsell/Cross Sell model for marketing
- Demand and sales forecasting Model

Mostly all predictive Models, Regression problems, Time series forecasting, Optimization problems using batch data can be fitted in this mechanism properly where elasticity/scalability/load balancing is not a concern.

9. SAMPLE USE CASE DOESN'T FITS THE ARCHITECTURE

There are few other types of use case that demands real time results or distributed processing for massive performance issues or high-end cognitive vision or NLP use cases or use cases that requires massive scalability/elasticity concern – cannot be suitable for this type of vanilla MLOps architecture. Few of the sample use cases are

- Realtime anomaly detection on telemetry data
- Abstractive text summary using Attention model
- SCM optimization at Walmart's scale
- Image segmentation using mask RCNN
- Document Intelligence using NLP

In those cases, we have no other choice other than using the so-called state of the art MLOps platforms mostly from all three major cloud providers like GCP – Kubeflow, Azure Devops or Databricks MLflow or AWS Sagemaker studio. The heavy computation pytorch, ONNX or TensorFlow type of computation doesn't fit properly with this type of architecture which actually built to cater the generic machine learning problems for small scale industry.

10. BENEFITS

- Absolutely low computation cost in terms of memory, tools, licenses and application platforms.
- Maintenance is easy because the underlying structure is easy, anyone can easily adopt the same.
- No need for any GPU/TPU type execution.
- Very much transparent and explainable solution, no hidden state for the entire pipeline.
- Very much use full for the novice data scientist because the simple Python, R, SQL and shell script will suffice 95% of the requirements.
- No cost for any third-party visualization.

- Can cater big chunk data using Modin pandas, Dask or Vaex (Python inbuilt capabilities)
- Use Python inbuilt parallelism using all cores of the server to distribute the processing capability without involving any spark or cloud capability.
- Easily integrated with any on-premises databases like SSMS, Oracle SQL, SQL server, Teradata.

11. LIMITATIONS

There are few limitations as well to use the architecture:

- Not useful for any TensorFlow, Pytorch or complex NLP task
- Not suitable for highly scalable solutions
- Cannot used for Real time requirements.
- Also, limitation where TBs of data load is involved.
- Cannot useful once data source be in HDFS, snowflake, RDS, Redshift, Athena, S3, EMR, BigQuery, BigTable, cosmos DB, Synapse or ADLS gen2.

12. CONCLUSIONS

Finally, the solutions have a solid background for lot of small-scale industries or even large scales those uses before 10 years where there was no existence of these state of art MLops platforms. In some of the cases its still very much effective and cost savings but in some cases, it has some limitations. This is a very custom solution and be adjusted based upon the requirements whereas we cannot adjust the architecture or workflow of any SOTA platforms for our own ease. Definitely lot of additional coding involves implementing the structure for an end-to-end solution, The main objective of sharing the architecture for those novice data scientists to test the framework and gain some in-depth knowledge of how MLOps is actually working rather than using a drag and Drop DAG platforms which is a grey box or black box mechanism doesn't actually explains the core processing at very ground level. I would really encourage junior practitioners to test the architectures on your own hands to get a better flavor of the core MLOps blocks.

REFERENCES

1. MLOps: Continuous delivery and automation pipelines in machine learning | Google Cloud

2. MLOps Principles (ml-ops.org)

3. Overview of MLOps - KDnuggets

4. Forrester: The future of machine learning is unstoppable (cloudera.com)

BIOGRAPHIES

Indranil Dutta is a Principal consultant and Lead Data scientist with more than 11 years of rich experience in Data science and Artificial Intelligence in various industries. His core competency is in delivering scalable Data science and AI projects in Big data and Cloud environments.